About Dataset Context The datasets provided include the players data for the Career Mode from FIFA 15 to FIFA 22 ("players_22.csv"). The data allows multiple comparisons for the same players across the last 8 version of the videogame.

Some ideas of possible analysis:

Historical comparison between Messi and Ronaldo (what skill attributes changed the most during time - compared to real-life stats);

Ideal budget to create a competitive team (at the level of top n teams in Europe) and at which point the budget does not allow to buy significantly better players for the 11-men lineup. An extra is the same comparison with the Potential attribute for the lineup instead of the Overall attribute;

Sample analysis of top n% players (e.g. top 5% of the player) to see if some important attributes as Agility or BallControl or Strength have been popular or not acroos the FIFA versions. An example would be seeing that the top 5% players of FIFA 20 are faster (higher Acceleration and Agility) compared to FIFA 15. The trend of attributes is also an important indication of how some attributes are necessary for players to win games (a version with more top 5% players with high BallControl stats would indicate that the game is more focused on the technique rather than the physicial aspect).

Content Every player available in FIFA 15, 16, 17, 18, 19, 20, 21, and also FIFA 22

100+ attributes

URL of the scraped players

URL of the uploaded player faces, club and nation logos

Player positions, with the role in the club and in the national team

Player attributes with statistics as Attacking, Skills, Defense, Mentality, GK Skills, etc.

Player personal data like Nationality, Club, DateOfBirth, Wage, Salary, etc.

Updates from previous FIFA 21 dataset are the following:

Inclusion of FIFA 22 data

Inclusion of all female players

Columns reorder - to increase readability

Removal of duplicate GK attribute fields

The field defending marking has been renamed defending marking awareness and includes both the marking (old attribute name - up to FIFA 19) and defensive awareness values (new attribute name - from FIFA 20)

All data from FIFA 15 was re-scraped, as one Kaggle user noted in this discussion that sofifa updated some historical player market values over time Data link : https://www.kaggle.com/datasets/stefanoleone992/fifa-22-complete-player-dataset (https://www.kaggle.com/datasets/stefanoleone992/fifa-22-complete-player-dataset)

K-clustering, also known as K-means clustering, is a popular unsupervised machine learning algorithm used for partitioning a given dataset into K clusters. The algorithm aims to group similar data points together and separate dissimilar data points based on their features or attributes.

Here's a step-by-step explanation of the K-means clustering algorithm:

1. Initialization: First, the number of clusters, K, is specified. The initial step involves randomly selecting K data points from the dataset as the initial centroids or cluster centers.
2. Assignment: Each data point in the dataset is assigned to the nearest centroid based on a distance metric, typically Euclidean distance. The distance between a data point and a centroid is calculated, and the data point is assigned to the cluster associated with the nearest centroid.
3. Update: After the assignment step, the centroids of the clusters are recomputed. The new centroids are obtained by taking the mean (average) of all the data points assigned to each cluster. This step ensures that the centroids are representative of the data points within each cluster.
4. Iteration: Steps 2 and 3 are repeated iteratively until convergence is reached. Convergence occurs when the centroids no longer change significantly or when a predetermined number of iterations is reached.
5. Finalization: Once the algorithm converges, the final K clusters are obtained. Each data point is associated with a specific cluster based on its nearest centroid.

K-means clustering is an iterative algorithm that aims to minimize the sum of squared distances between data points and their respective cluster centroids. However, it is important to note that K-means clustering is sensitive to the initial selection of centroids and may converge to different results based on the initial random assignment.

This algorithm has various applications, such as customer segmentation, image compression, anomaly detection, and document clustering, among others. It is a widely used technique in data mining and exploratory data analysis.

In [ ]:
```
1
```

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
players = pd.read_csv('player_folder/players_22.csv', low_memory=False)

```

In [3]:
```
players.head()
```

Out[3]:

	sofifa_id	player_url	short_name	long_name	player_positions	overall	potential	value_eur	wage_e
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	RW, ST, CF	93	93	78000000.0	32000(
1	188545	https://sofifa.com/player/188545/robert-lewand...	R. Lewandowski	Robert Lewandowski	ST	92	92	119500000.0	27000(
2	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	ST, LW	91	91	45000000.0	27000(
3	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neymar da Silva Santos Júnior	LW, CAM	91	91	129000000.0	27000(
4	192985	https://sofifa.com/player/192985/kevin-de-bruy...	K. De Bruyne	Kevin De Bruyne	CM, CAM	91	91	125500000.0	35000(

5 rows × 110 columns

In [4]:
```
players.columns
```

Out[4]:
```
Index(['sofifa_id', 'player_url', 'short_name', 'long_name',
       'player_positions', 'overall', 'potential', 'value_eur', 'wage_eur',
       'age',
       ...
       'lcb', 'cb', 'rcb', 'rb', 'gk', 'player_face_url', 'club_logo_url',
       'club_flag_url', 'nation_logo_url', 'nation_flag_url'],
      dtype='object', length=110)
```

In [5]:
```
players.shape
```

Out[5]:
```
(19239, 110)
```

In [6]:
```
players.isna().sum()
```

Out[6]:
```
sofifa_id              0
player_url             0
short_name             0
long_name              0
player_positions       0
                     ...
player_face_url        0
club_logo_url         61
club_flag_url         61
nation_logo_url     18480
nation_flag_url         0
Length: 110, dtype: int64
```

In [7]:
```
players.isnull().sum().shape
```

Out[7]:
```
(110,)
```

In [8]:
```
# features we will use for our clustering project

features = ['overall', 'potential', 'value_eur', 'wage_eur','age',]
```

In [9]:
```
features
```

Out[9]:
```
['overall', 'potential', 'value_eur', 'wage_eur', 'age']
```

In [10]:
```
players = players.dropna(subset = features)
```

In [11]:
```
1 players
```

Out[11]:

	sofifa_id	player_url	short_name	long_name	player_positions	overall	potential	value
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	RW, ST, CF	93	93	78000(
1	188545	https://sofifa.com/player/188545/robert-lewand...	R. Lewandowski	Robert Lewandowski	ST	92	92	119500(
2	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	ST, LW	91	91	45000(
3	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neymar da Silva Santos Júnior	LW, CAM	91	91	129000(
4	192985	https://sofifa.com/player/192985/kevin-de-bruy...	K. De Bruyne	Kevin De Bruyne	CM, CAM	91	91	125500(
...	...	...	...	...	...	...	...	
19234	261962	https://sofifa.com/player/261962/defu-song/220002	Song Defu	宋德福	CDM	47	52	70(
19235	262040	https://sofifa.com/player/262040/caoimhin-port...	C. Porter	Caoimhin Porter	CM	47	59	110(
19236	262760	https://sofifa.com/player/262760/nathan-logue/...	N. Logue	Nathan Logue-Cunningham	CM	47	55	100(
19237	262820	https://sofifa.com/player/262820/luke-rudden/2...	L. Rudden	Luke Rudden	ST	47	60	110(
19238	264540	https://sofifa.com/player/264540/emanuel-lalch...	E. Lalchhanchhuaha	Emanuel Lalchhanchhuaha	CAM	47	60	110(

19165 rows × 110 columns

In [12]:
```
1 df = players[features].copy()
```

In [13]:
```
1 df
```

Out[13]:

	overall	potential	value_eur	wage_eur	age
0	93	93	78000000.0	320000.0	34
1	92	92	119500000.0	270000.0	32
2	91	91	45000000.0	270000.0	36
3	91	91	129000000.0	270000.0	29
4	91	91	125500000.0	350000.0	30
...	...	...	...	...	...
19234	47	52	70000.0	1000.0	22
19235	47	59	110000.0	500.0	19
19236	47	55	100000.0	500.0	21
19237	47	60	110000.0	500.0	19
19238	47	60	110000.0	500.0	19

19165 rows × 5 columns

In [14]:
```
1 df.isna().sum()
```

Out[14]:
```
overall      0
potential    0
value_eur    0
wage_eur     0
age          0
dtype: int64
```

1. scale the data
2. Initialize random centroid
3. label each data point
4. Update centroid
5. Reapeat step 3 and 4 until centroid stop changing

In [15]:
```
# scaling our data

df = (df - df.min()) / (df.max() - df.min()) * 9 + 1
```

In [16]:
```
df
```

Out[16]:

	overall	potential	value_eur	wage_eur	age
0	10.000000	9.608696	4.618307	9.227468	7.000000
1	9.804348	9.413043	6.543654	7.939914	6.333333
2	9.608696	9.217391	3.087308	7.939914	7.666667
3	9.608696	9.217391	6.984396	7.939914	5.333333
4	9.608696	9.217391	6.822018	10.000000	5.666667
...	...	...	...	...	...
19234	1.000000	1.586957	1.002830	1.012876	3.000000
19235	1.000000	2.956522	1.004686	1.000000	2.000000
19236	1.000000	2.173913	1.004222	1.000000	2.666667
19237	1.000000	3.152174	1.004686	1.000000	2.000000
19238	1.000000	3.152174	1.004686	1.000000	2.000000

19165 rows × 5 columns

In [17]:
```
1 df.describe()
```

Out[17]:

	overall	potential	value_eur	wage_eur	age
count	19165.000000	19165.000000	19165.000000	19165.000000	19165.000000
mean	4.670472	5.319998	1.131826	1.219443	4.063345
std	1.346635	1.191076	0.353229	0.501528	1.575838
min	1.000000	1.000000	1.000000	1.000000	1.000000
25%	3.739130	4.521739	1.021620	1.012876	2.666667
50%	4.717391	5.304348	1.044817	1.064378	4.000000
75%	5.500000	6.086957	1.092370	1.193133	5.333333
max	10.000000	10.000000	10.000000	10.000000	10.000000

from here we can see that our min value is 1 and max is 10

In [18]:
```
1 # geting our centroid of each columns
2
3 centroid = df.apply(lambda x: float(x.sample()))
```

In [19]:
```
1 centroid
```

Out[19]:
```
overall      3.152174
potential    6.673913
value_eur    1.032058
wage_eur     1.141631
age          2.666667
dtype: float64
```

In [20]:
```
# initializing random centroid
def random_centroid(df, k):
    centroids = []  # Corrected variable name
    for i in range(k):
        centroid = df.apply(lambda x: float(x.sample()))
        centroids.append(centroid)
    return pd.concat(centroids, axis=1)

centroids = random_centroid(df, 5)

```

This is the definition of the random_centroid function. It takes two arguments: df and k. df is expected to be a DataFrame object, and k represents the number of centroids you want to generate.

Inside the function, an empty list named centroids is created to store the centroid values. The function then enters a loop that iterates k times. In each iteration, the code generates a single centroid by sampling a random value from each column of the DataFrame df. This is done using the apply method along with a lambda function that converts the sampled value to a float.

The resulting centroid is stored in the centroid variable, and then appended to the centroids list. This process repeats k times, generating k centroids.

Finally, the function returns the concatenated centroids using the pd.concat function, with axis=1 indicating that the concatenation should be performed along the columns.

In [21]:
```
centroids = random_centroid(df, 5)
```

In [22]:
```
centroids
```

Out[22]:

	0	1	2	3	4
overall	6.086957	1.978261	6.673913	3.934783	6.282609
potential	7.456522	5.304348	6.478261	6.478261	8.043478
value_eur	1.007933	1.050616	1.025099	1.029738	1.649097
wage_eur	1.038627	1.167382	1.038627	1.064378	1.090129
age	4.333333	4.333333	2.666667	4.333333	5.333333

In [23]:
```
1 centroids.shape
```

Out[23]: (5, 5)

In [24]:
```
1 # finding the distance between the main data and the centroid
2
3 distances = centroids.apply(lambda x: np.sqrt(((df - x) ** 2).sum(axis=1)))
```

In [25]:
```
1 distances
```

Out[25]:

	0	1	2	3	4
0	10.351173	12.949140	10.936568	11.541727	9.699442
1	9.995947	12.576553	10.485191	11.168752	9.281950
2	8.863746	11.604390	9.643457	10.115030	8.177615
3	9.992456	12.474469	10.313481	11.109774	9.371547
4	11.462586	13.661531	11.790940	12.444667	10.894813
...	...	...	...	...	...
19234	7.880824	4.071846	7.498694	5.858239	8.686812
19235	7.181440	3.455988	6.711355	5.144396	8.081929
19236	7.520795	3.683010	7.121982	5.470186	8.360194
19237	7.060488	3.326172	6.610786	5.012486	7.960233
19238	7.060488	3.326172	6.610786	5.012486	7.960233

19165 rows × 5 columns

In [26]:
```
# find the index of the minimum value of each roll
distances.idxmin(axis= 1)
```

Out[26]:
```
0        4
1        4
2        4
3        4
4        4
        ..
19234    1
19235    1
19236    1
19237    1
19238    1
Length: 19165, dtype: int64
```

In [27]:
```
# function to get label for each data point

def get_label(df, centroids):
    distances = centroids.apply(lambda x: np.sqrt(((df - x) ** 2).sum(axis=1)))
    return distances.idxmin(axis= 1)
```

Explanation:

The get_label function takes two parameters: df, which represents the dataset containing data points, and centroids, which represents the centroids to which the data points are compared.

Inside the function, the apply method is used on the centroids DataFrame. This applies a lambda function to each centroid, which calculates the Euclidean distance between each data point in the df DataFrame and the centroid. The resulting distances are stored in the distances DataFrame.

Finally, the idxmin method is used to find the index of the centroid with the minimum distance for each data point. This index represents the label assigned to the data point. The axis=1 parameter indicates that the minimum index should be found along the columns.

Possible correction:

The code seems to be missing an import statement for the numpy module (import numpy as np). Without the import, the code will raise an error when trying to use np.sqrt to calculate the square root. To correct it, you can add the following import statement at the beginning of the code:

In [28]:
```
1 labels = get_label(df, centroids)
```

In [29]:
```
1 labels
```

Out[29]:
```
0        4
1        4
2        4
3        4
4        4
        ..
19234    1
19235    1
19236    1
19237    1
19238    1
Length: 19165, dtype: int64
```

In [30]:
```
1 labels.value_counts()
```

Out[30]:
```
3    9548
1    4243
0    2302
2    1725
4    1347
dtype: int64
```

In [31]:
```
1 labels.shape
```

Out[31]:
```
(19165,)
```

In [32]:
```
# finding the geometric mean
df.groupby(labels).apply(lambda x: np.exp(np.log(x).mean())).T
```

Out[32]:

	0	1	2	3	4
overall	6.199018	2.870070	5.540072	4.514491	6.938244
potential	6.247619	4.001329	6.818180	5.103370	6.627945
value_eur	1.247643	1.013268	1.215895	1.046322	1.553220
wage_eur	1.475477	1.021052	1.287564	1.078919	1.891650
age	4.524770	3.134790	2.667058	3.858587	6.011704

In [33]:
```
# function to update centroid
def new_centroid(df, labels, k):
    return df.groupby(labels).apply(lambda x: np.exp(np.log(x).mean())).T
```

The code performs the following steps:

df.groupby(labels): This line groups the DataFrame df based on the specified labels. It returns a GroupBy object.

.apply(lambda x: np.exp(np.log(x).mean())): This line applies a lambda function to each group in the GroupBy object. The lambda function takes each group (x), calculates the logarithm of each value, computes the mean of the logarithmic values, and then exponentiates the mean. Essentially, it calculates the geometric mean of each group. This operation is done using NumPy functions np.log() and np.exp().

.T: This line transposes the resulting DataFrame. The resulting DataFrame will have the updated centroids, where each row represents a centroid and each column represents a feature.

Finally, the function new_centroid returns the transposed DataFrame containing the updated centroids.

It's important to note that the code assumes that the DataFrame df has numerical values, as the calculations involve logarithms and exponentiation. Additionally, the code uses NumPy (np) for mathematical operations. Therefore, you need to import the NumPy library (import numpy as np) for the code to work correctly.

In [34]:
```
from sklearn.decomposition import PCA #principle component analysis
from IPython.display import clear_output
```

In [35]:
```
# function to plot our clusters
def plot_clusters(df, labels, centroids, iteration):
    pca = PCA(n_components=2)
    data_2d = pca.fit_transform(df)
    centroids_2d = pca.transform(centroids.T)
    clear_output(wait=True)
    plt.title(f'iteration {iteration}')
    plt.scatter(x=data_2d[:, 0], y=data_2d[:, 1], c=labels)
    plt.scatter(x=centroids_2d[:, 0], y=centroids_2d[:, 1])
    plt.show()

```

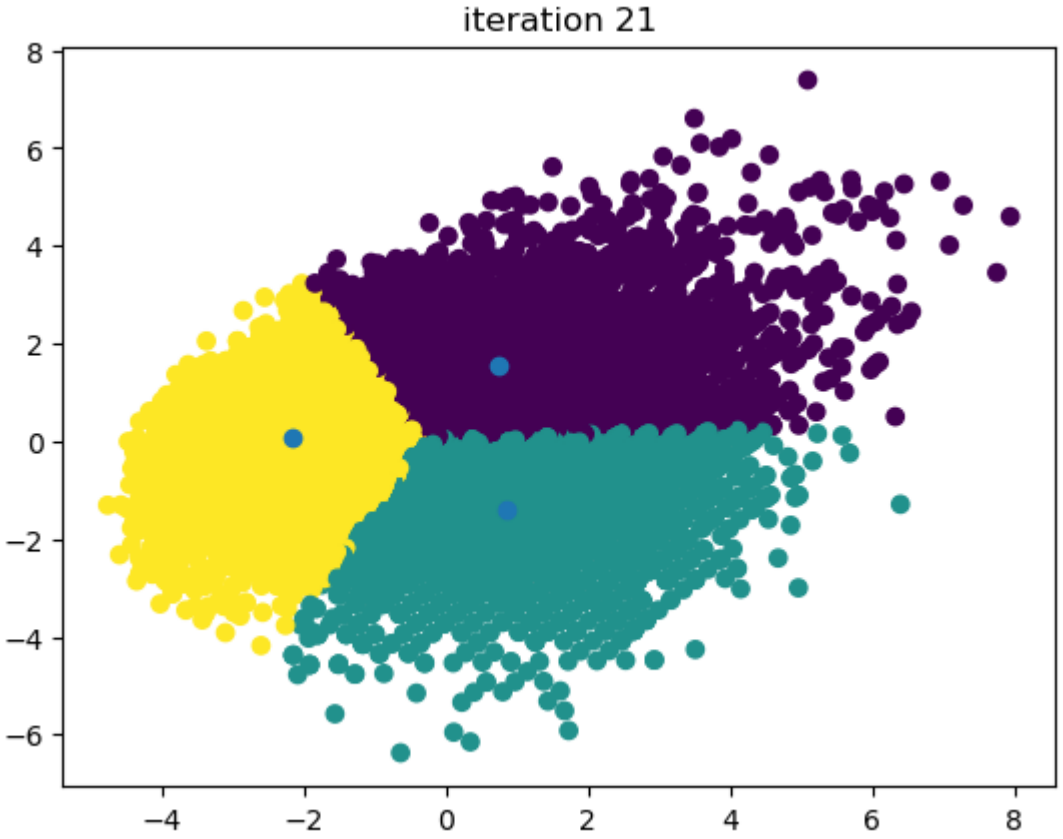
In [41]:
```
max_iter = 100
k = 3 # number clusters

centroids = random_centroid(df, k)
old_centroids = pd.DataFrame()
iteration = 1

while iteration < max_iter and not centroids.equals(old_centroids):
    old_centroids = centroids

    labels = get_label(df, centroids)
    centroids = new_centroid(df, labels, k)
    plot_clusters(df, labels, centroids, iteration)
    iteration += 1

```



Certainly! Let's break down the code step by step:

max_iter = 100: This line sets the maximum number of iterations for the k-means algorithm. It specifies the maximum number of times the algorithm will iterate to find the optimal cluster centroids.

k = 3: This line defines the number of clusters you want to create. In this case, it is set to 3, meaning the algorithm will try to identify and assign data points to three different clusters. later changed to 5

centroids = random_centroid(df, k): This line initializes the centroids by calling the random_centroid function. It takes in the dataframe df and the number of clusters k as parameters and returns randomly selected data points as the initial centroid locations.

old_centroids = pd.DataFrame(): This line initializes an empty DataFrame called old_centroids to store the previous centroids.

iteration = 1: This line initializes the iteration counter to 1. It keeps track of the current iteration number.

The while loop: This loop iterates until either the maximum number of iterations is reached (iteration < max_iter) or the centroids stop changing (not centroids.equals(old_centroids)).

old_centroids = centroids: This line updates the old_centroids variable to store the previous centroid locations before the new centroids are computed.

labels = get_label(df, centroids): This line assigns labels to the data points based on their distances to the centroids. The get_label function takes the dataframe df and the current centroids as parameters and returns a list of labels corresponding to each data point.

centroids = new_centroids(df, labels, k): This line updates the centroids based on the current labels. The new_centroids function takes the dataframe df, the labels, and the number of clusters k as parameters and computes new centroid locations based on the mean of the data points assigned to each cluster.

plot_clusters(df, labels, centroids, iteration): This line calls the plot_clusters function to visualize the clusters and their centroids at the current iteration. It takes the dataframe df, the labels, the current centroids, and the iteration number as parameters.

iteration += 1: This line increments the iteration counter by 1 to move to the next iteration.

The loop continues until either the maximum number of iterations is reached or the centroids no longer change, indicating convergence. The plot_clusters function is called at each iteration to visualize the clusters and their centroids.

In [42]:
```
1 centroids
```

Out[42]:

	0	1	2
overall	5.807503	4.781960	3.205672
potential	6.497870	4.506813	4.930905
value_eur	1.285685	1.044909	1.026655
wage_eur	1.420500	1.118498	1.028564
age	3.598215	5.467648	2.514741

In [43]:
```
# checking players each cluster is refering to:

players[labels == 0]
```

Out[43]:

	sofifa_id	player_url	short_name	long_name	player_positions	overall	potential	value_eur	w
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	RW, ST, CF	93	93	78000000.0	3
1	188545	https://sofifa.com/player/188545/robert-lewand...	R. Lewandowski	Robert Lewandowski	ST	92	92	119500000.0	2
2	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	ST, LW	91	91	45000000.0	2
3	190871	https://sofifa.com/player/190871/neymar-da-sil...	Neymar Jr	Neymar da Silva Santos Júnior	LW, CAM	91	91	129000000.0	2
4	192985	https://sofifa.com/player/192985/kevin-de-bruy...	K. De Bruyne	Kevin De Bruyne	CM, CAM	91	91	125500000.0	3
...	...	...	...	...	...	...	...	...	
13245	261025	https://sofifa.com/player/261025/dane-scarlett...	D. Scarlett	Dane Pharrell Scarlett	ST	63	86	1500000.0	
13256	261374	https://sofifa.com/player/261374/lewis-bate/22...	L. Bate	Lewis Bate	CM, CDM	63	83	1300000.0	
13351	264110	https://sofifa.com/player/264110/javier-llabre...	Llabrés	Javier Llabrés Exposito	LM, LW, RW	63	81	1200000.0	
14144	258171	https://sofifa.com/player/258171/hannibal-mejb...	H. Mejbri	Hannibal Mejbri	CAM, CM	62	84	1300000.0	
14349	263620	https://sofifa.com/player/263620/romeo-lavia/2...	R. Lavia	Romeo Lavia	CDM	62	85	1200000.0	

5765 rows × 110 columns

In [46]:
```
players[labels == 0][['short_name'] + features]
```

Out[46]:

	short_name	overall	potential	value_eur	wage_eur	age
0	L. Messi	93	93	78000000.0	320000.0	34
1	R. Lewandowski	92	92	119500000.0	270000.0	32
2	Cristiano Ronaldo	91	91	45000000.0	270000.0	36
3	Neymar Jr	91	91	129000000.0	270000.0	29
4	K. De Bruyne	91	91	125500000.0	350000.0	30
...	...	...	...	...	...	...
13245	D. Scarlett	63	86	1500000.0	3000.0	17
13256	L. Bate	63	83	1300000.0	5000.0	18
13351	Llabrés	63	81	1200000.0	3000.0	19
14144	H. Mejbri	62	84	1300000.0	6000.0	18
14349	R. Lavia	62	85	1200000.0	700.0	17

5765 rows × 6 columns

In [47]:
```
# let's compare it to the implementation in sklearn
from sklearn.cluster import KMeans
```

In [49]:
```
kmeans = KMeans(3)
kmeans.fit(df)
```

Out[49]:
```
KMeans(n_clusters=3)
```

In [50]:
```
# to get the centroid
centroids = kmeans.cluster_centers_

```

In [55]:
```
1 centroids
```

Out[55]:
```
array([[3.59066541, 6.22188766, 4.80098383],
       [5.20309523, 6.61898461, 4.50617117],
       [1.03552898, 1.41141529, 1.04007618],
       [1.03959764, 1.65311812, 1.11286858],
       [2.70531214, 4.13416149, 5.6039709 ]])
```

In [57]:
```
1 pd.DataFrame(centroids, columns=features).T
```

Out[57]:

	0	1	2
overall	3.590665	6.221888	4.800984
potential	5.203095	6.618985	4.506171
value_eur	1.035529	1.411415	1.040076
wage_eur	1.039598	1.653118	1.112869
age	2.705312	4.134161	5.603971

In [ ]:
```
1
```